Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»**
Powershell terminal

Виконав:
Студент групи ІА-22
Парій І. Р.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Singleton», «Iterator», «Proxy», «State» та «Strategy», дослідити їхні принципи роботи та навчитися використовувати для створення гнучкого та масштабованого програмного забезпечення.

Варіант
..8 Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server)
Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

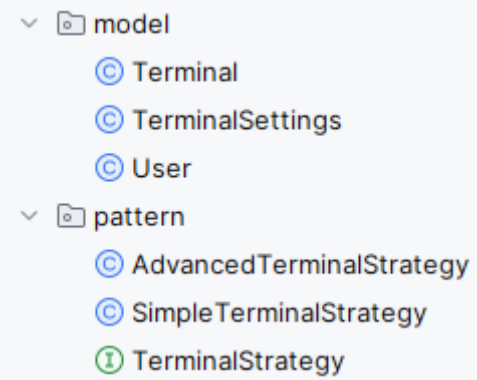

Рис. 1 — Структура проекту

У процесі виконання лабораторної роботи було створено 4 класи, які забезпечують реалізацію функціоналу PowerShell термінал (рис. 1). Наведемо детальний опис кожного з цих класів:

1. Інтерфейс TerminalStrategy
   Цей інтерфейс задає контракт для стратегій виконання команд у терміналі. Він має метод executeCommand(String command), який

визначає, як саме слід обробляти команду. Інші класи, які реалізують цей інтерфейс, забезпечують конкретну логіку виконання команд.

2. Клас AdvancedTerminalStrategy
   Реалізація інтерфейсу TerminalStrategy, яка моделює виконання складних команд у терміналі. Його метод executeCommand додає специфічний опис до команди, щоб показати, що вона є складною. Використовується для ситуацій, коли термінал повинен виконувати більш просунуті операції.
3. Клас SimpleTerminalStrategy
   Реалізація інтерфейсу TerminalStrategy, яка відповідає за виконання простих команд. Її метод executeCommand маркує команду як просту. Цей клас використовується для базових операцій у терміналі, які не потребують складних алгоритмів чи додаткової обробки.
4. Клас Terminal
   Основний клас, який представляє термінал і працює зі стратегіями для виконання команд.
   - Має поле TerminalStrategy terminalStrategy, яке зберігає поточну стратегію виконання.
   - Метод executeCommand(String command) делегує виконання команди поточній стратегії.
   - Конструктор із параметром дозволяє задавати початкову стратегію терміналу.
   - Метод setTerminalStrategy дає змогу змінювати стратегію виконання команд під час роботи програми.

   Цей клас забезпечує гнучкість і підтримує патерн "Стратегія", дозволяючи легко змінювати логіку виконання команд.

2. Реалізувати один з розглянутих шаблонів за обраною темою

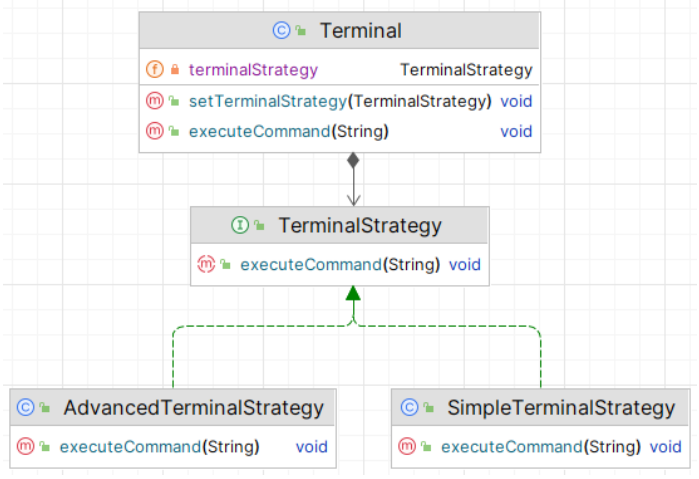

Рис. 2 — Діаграма класів

У даному проєкті патерн Strategy був використаний для забезпечення гнучкості в реалізації різних способів виконання команд PowerShell у терміналі. Ключова ідея полягала у виділенні логіки виконання команд в окремі стратегії, які можна легко змінювати під час роботи програми.

Як це реалізовано:

1. Інтерфейс TerminalStrategy
   - Слугує контрактом для всіх стратегій виконання команд.
   - Забезпечує єдиний метод executeCommand, що дозволяє стандартизувати спосіб роботи з командами.
2. Реалізації стратегій
   - SimpleTerminalStrategy: Обробляє базові команди PowerShell, наприклад, запуск скриптів або виконання простих операцій (наприклад, обчислень).
   - AdvancedTerminalStrategy: Підтримує складніші сценарії, такі як виконання великого обсягу команд або обробка команд із багаторівневою логікою (наприклад, управління виконуваними файлами чи інтеграція з іншими системами).
3. Клас Terminal
   - Інкапсулює основний функціонал термінала, включаючи можливість обрати та динамічно змінювати стратегію виконання команд.
   - Завдяки делегуванню команд через інтерфейс TerminalStrategy забезпечується гнучкість і легка масштабованість.

Проблеми, які вирішує:

1. Розширюваність функціоналу
   - Додавання нових способів виконання команд не потребує змін у класі Terminal, лише створення нових реалізацій інтерфейсу TerminalStrategy. Це відповідає принципу відкритості/закритості (OCP).
2. Гнучкість налаштування
   - У різних сценаріях роботи (простий термінал чи виконання складних скриптів) можна динамічно змінювати стратегію, адаптуючи поведінку термінала до конкретних потреб користувача.
3. Модульність
   - Логіка виконання команд винесена в окремі класи, що робить код легшим для тестування, підтримки й масштабування.

Переваги використання патерна Strategy:

1. Динамічна зміна поведінки: користувач може перемикатися між різними стратегіями виконання команд без необхідності перезапуску термінала або зміни коду.
2. Проста інтеграція нових функцій: додавання підтримки нових типів команд або змін у способах їх виконання не зачіпає існуючий код термінала.
3. Гнучкість архітектури: термінал легко адаптується до потреб різних користувачів, наприклад, додавання можливості роботи з кількома вікнами або вкладками не впливає на модуль виконання команд.

Застосування

```
@RestController
@RequestMapping("/terminal")
@AllArgsConstructor
public class TerminalController {
    private final Terminal terminal;

    @GetMapping("/execute/{command}")
    public String executeCommand(@PathVariable String command) {
        terminal.executeCommand(command);
        return "Command executed: " + command;
    }
}
```

Рис. 3 — Застосування патерну

У поданому коді клас TerminalController забезпечує API для виконання команд PowerShell через термінал у проекті. Впровадження патерну Strategy дозволяє динамічно змінювати логіку виконання команд, що інкапсулюється в класі Terminal. Завдяки цьому термінал підтримує різні стратегії виконання, наприклад, базову (SimpleTerminalStrategy) для простих команд і розширену (AdvancedTerminalStrategy) для складних сценаріїв. Це забезпечує гнучкість та масштабованість системи.

Висновки:

Завдяки реалізації патерна Strategy, термінал PowerShell отримав масштабовану, модульну та гнучку архітектуру. Це дозволяє легко впроваджувати нові функції, зберігаючи високий рівень абстракції та відповідність принципам SOLID.